

Einführung MQTT-Broker – Installation und Konfiguration

MQTT ist ein leichtgewichtiges Kommunikationsprotokoll, das häufig in der Heimautomatisierung, im IoT (Internet der Dinge) und bei einfachen Nachrichtensystemen verwendet wird. Die grundlegende Architektur besteht dabei aus einem zentralen **Broker („Server“)** und einem oder mehreren **Clients**, welche als **Publisher** und/oder **Subscriber** agieren können.

Insbesondere für die Bearbeitung der Aufgaben im **Kapitel 1.4 und 1.5** sowie ggf. für die Umsetzung eigener **Projekte** wird ein MQTT-Broker benötigt, mit welchem sich die von den Schülern entwickelten Clients verbinden können. Diese Anleitung beschreibt, wie ein solcher Broker auf verschiedene Weise konfiguriert werden kann, sodass die Kommunikation wahlweise mit/ohne Authentifizierung und mit/ohne Verschlüsselung erfolgt.

Voraussetzungen und Installation

Im folgenden wird die Einrichtung eines MQTT-Brokers auf Basis von **Mosquitto** beschrieben, welches als Open-Source-Software kostenlos aus dem Internet heruntergeladen werden kann und sowohl für Linux, Windows, als auch MacOS verfügbar ist.

Linux

Bei vielen Linux-Distributionen kann Mosquitto direkt über den Paketmanager installiert werden. Dies gilt insbesondere auch für „Raspberry Pi OS“, bei dem die Installation mittels des Terminalbefehls

```
sudo apt install mosquitto mosquitto-clients
```

möglich ist. Für die Erstellung von SSL-Zertifikaten sollte zusätzlich das OpenSSL-Paket installiert sein. Falls dies nicht ohnehin bereits installiert wurde, kann es via Paketmanager installiert werden; bei Raspberry Pi OS z. B. mittels

```
sudo apt install openssl
```

Sämtliche Mosquitto-Konfigurationsdateien befinden sich nach der Installation im Verzeichnis `/etc/mosquitto/`.

Bei systemd-basierten Systemen (gilt für die meisten modernen Linux-Distributionen) kann der Mosquitto-Dienst mittels des Terminalbefehls `systemctl` verwaltet werden. Zu erwähnen ist hierbei insbesondere:

<code>sudo systemctl start mosquitto</code>	Startet den Dienst
<code>sudo systemctl stop mosquitto</code>	Beendet den Dienst
<code>sudo systemctl status mosquitto</code>	Gibt Statusinformationen zum Dienst im Terminal aus
<code>sudo systemctl enable mosquitto</code>	Sorgt dafür, dass der Dienst automatisch gestartet wird
<code>sudo systemctl disable mosquitto</code>	Sorgt dafür, dass der Dienst nicht automatisch gestartet wird.

Windows

Für Windows können fertige Installationspakete für Mosquitto und OpenSSL heruntergeladen werden. Der Installationsassistent kann Mosquitto dabei auch gleich automatisch als Dienst einrichten (Admin-Rechte erforderlich!). Die Programmdateien inklusive der Konfigurationsdateien liegen dann (sofern im Installationsassistent nichts anderes angegeben wurde) unter `C:\Programme\mosquitto\`. Alternativ kann die Programmdatei `mosquitto.exe` aber auch ohne Installation direkt aus dem Terminal gestartet werden. Durch den Parameter `-v` bleibt das Programm im Vordergrund und gibt Statusinformationen im Terminal aus. Durch `-c` kann der Pfad zur Konfigurationsdatei festgelegt werden. Ein Start des Brokers kann entsprechend so aussehen:

```
mosquitto.exe -v -c mosquitto.conf
```

MacOS

Unter MacOS erfolgt die Installation von Mosquitto und ggf. OpenSSL mittels `homebrew`. Die Mosquitto-Konfigurationsdateien befinden sich (sofern nichts anderes angegeben wurde) anschließend unter `/usr/local/etc/mosquitto/`. Mittels des Terminalbefehls

```
brew services start mosquitto
```

kann Mosquitto analog zum Vorgehen bei Linux als Dienst gestartet werden.

Konfiguration

Zentral für die Konfiguration von Mosquitto ist die Datei `mosquitto.conf`. Wird Mosquitto neu installiert, liegt der Installation in der Regel eine Beispielkonfigurationsdatei bei, in welcher alle möglichen Optionen per Kommentar innerhalb der Konfigurationsdatei erläutert werden.

Für die Aufgaben im Schulbuch sowie ggf. die Projektarbeit wird jedoch in der Regel nur ein Bruchteil dieser Konfigurationsparameter benötigt. Diesem Dokument sind daher verschiedene stark vereinfachte Konfigurationsdateien beigelegt, welche sich auf die jeweils wesentlichen Parameter beschränken und diese mit deutschen Kommentaren erklären:

- `mosquitto_einfach.conf` MQTT-Broker ohne Authentifizierung / Verschlüsselung
- `mosquitto_auth.conf` MQTT-Broker mit Authentifizierung / ohne Verschlüsselung
- `mosquitto_auth_ssl.conf` MQTT-Broker mit Authentifizierung / mit Verschlüsselung
- `mosquitto_auth_ssl_acl.conf` ...wie oben, zusätzlich werden ACLs definiert

Um diese Konfigurationen zu verwenden, muss die vom Dienst verwendete `mosquitto.conf`-Datei durch eine der oben genannten Dateien ersetzt werden und der Mosquitto-Dienst anschließend neu gestartet werden.

Erstellen von Passworthashes (benötigt für Broker mit Authentifizierung)

Soll ein Broker mit Benutzerauthentifizierung eingerichtet werden, ist hierfür eine separate Datei zu erstellen, in welcher Tupel aus Benutzername und zugehörigem Passworthash aufgelistet sind. Mosquitto bringt zum Erstellen einer solchen Datei das Kommandozeilentool `mosquitto_passwd` mit.

Ist noch keine Benutzerdatei vorhanden kann sie mittels `-c` Parameter erstellt werden:

```
mosquitto_passwd -c pfad/zur/benutzerdatei benutzername
```

Durch das Weglassen des `-c` Parameters kann eine bereits bestehende Benutzerdatei um weitere Einträge erweitert werden:

```
mosquitto_passwd pfad/zur/benutzerdatei benutzername2
```

Erstellen von Zertifikaten (benötigt für Broker mit SSL-Verschlüsselung)

Bei einem Broker mit SSL-Verschlüsselung findet neben der eigentlichen Verschlüsselung auch eine Geräteauthentifizierung mittels Zertifikaten statt. Gängig ist hierbei, dass der Client ein vom Broker vorgelegtes Zertifikat überprüft und so (noch vor der Übermittlung von Benutzername/Passwort) sicherstellt, dass er tatsächlich mit dem gewünschten Broker (und nicht einem „Man in the middle“) kommuniziert. Damit dies möglich ist, muss das Serverzertifikat, ein passendes direkt übergeordnetes CA-Zertifikat, oder eine entsprechende vollständige Zertifikatskette dem Client vorliegen.

Optional kann die Authentifizierung auch in die andere Richtung erfolgen bzw. vom Broker erzwungen werden: In diesem Fall muss zusätzlich auch der Client ein vertrauenswürdigen Zertifikat vorlegen können, um sich mit dem Broker zu verbinden.

Im folgenden wird beschrieben, wie mittels OpenSSL Zertifikate und dazu passende Schlüssel für eine gemeinsame CA, den Broker und einen Client erzeugt werden können. Bei mehreren Clients kann das Vorgehen für den Client beliebig oft wiederholt werden.

Wichtig: Im Rahmen der Schritte 2, 4 und 7 im folgenden Ablauf müssen diverse Angaben zum Zertifikat gemacht werden. Diese erscheinen später im Zertifikat. Für Test-/Übungszwecke sind die meisten Angaben egal. Für eine erfolgreiche Validierung des Serverzertifikats muss im Zertifikat des Servers unter „**Common Name**“ der FQDN oder die IP-Adresse des Brokers angegeben sein. Ist dies nicht der Fall, schlägt die Zertifikatsprüfung und somit auch die verschlüsselte Verbindung zum Broker fehl!

Hinweis: Soll ein Zertifikat für mehrere Adressen bzw. FQDNs gültig sein, ist dies mittels der sog. „Subject Alternative Name“-Erweiterung möglich. Hierzu muss bei der Erstellung des CSR (Schritte 4 bzw. 7) jeweils so oft wie benötigt folgender Parameter an den unten genannten Befehl angehängt werden:

-addext "subjectAltName = DNS:beispielsubdomain.beispieldomain.de"

Damit die Subject-Alternative-Name-Einträge in das Zertifikat übernommen werden, muss zusätzlich bei der Zertifikatsausstellung (Schritte 5 bzw. 8) folgender Parameter ergänzt werden:

-copy_extensions copy

1. Schlüssel für CA-Zertifikat erzeugen:

```
openssl genrsa -out ca.key 2048
```

2. CA-Zertifikat mit einer Gültigkeitsdauer von 10 Jahren erzeugen:

```
openssl req -new -x509 -days 3650 -key ca.key -out ca.crt
```

3. Schlüssel für das Serverzertifikat erzeugen:

```
openssl genrsa -out server.key 2048
```

4. Antrag (CSR) für das Serverzertifikat erstellen:

```
openssl req -new -key server.key -out server.csr
```

5. Serverzertifikat mit Gültigkeitsdauer von 10 Jahren durch CA ausstellen:

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 3650
```

6. Schlüssel für das Clientzertifikat erzeugen:

```
openssl genrsa -out client.key 2048
```

7. Antrag (CSR) für das Clientzertifikat erstellen:

```
openssl req -new -key client.key -out client.csr
```

8. Clientzertifikat mit Gültigkeitsdauer von 10 Jahren durch CA ausstellen:

```
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 3650
```

Der Broker benötigt von den erstellten Dateien: ca.crt, server.crt, server.key

Der Client benötigt ebenfalls ca.crt und ggf. client.crt / client.key bei aktivierter Client-Authentifizierung

Test / Debugging

Zusätzlich zum Broker werden bei der Installation von Mosquitto (bzw. unter Linux ggf. separat im Paket mosquitto-clients) zwei Hilfsprogramme mit installiert, mit denen einfache Publisher / Subscriber im Terminal simuliert werden können.

Mit dem Befehl

```
mosquitto_pub -h BROKER -t TOPIC -m PAYLOAD
```

kann eine einfache Nachricht publiziert werden, wobei BROKER durch den FQDN oder die IP-Adresse des Brokers und TOPIC/PAYLOAD durch den gewünschten Kanal und Nachrichteninhalt ersetzt werden müssen. Letztere müssen ggf. in Anführungszeichen gesetzt werden, sofern sie Steuerzeichen der verwendeten Terminal Shell enthalten.

Mit dem Befehl

```
mosquitto_sub -h BROKER -v -t TOPIC
```

kann auf gleiche Weise ein Kanal abonniert werden. Der Parameter -v sorgt zudem dafür, dass bei der Ausgabe empfangener Nachrichten der betroffene Kanal mit ausgegeben wird (hilfreich, falls mehrere Kanäle abonniert werden).

Bei beiden Befehlen können bei Bedarf die folgenden weiteren Parameter ergänzt werden:

Benutzerauthentifizierung:

```
-u Benutzername -P Passwort
```

Verschlüsselte Verbindung auf Port 8883 (SSL) und Überprüfung des Broker-Zertifikates anhand des angegebenen CA-Zertifikates:

```
-p 8883 --cafile ca.crt
```

Wie oben, zusätzlich Client-Authentifizierung mittels angegebenem Clientzertifikat:

```
-p 8883 --cafile ca.crt --cert client.crt --key client.key
```